TARDEC

---TECHNICAL REPORT---

No. 14386

By: Wesley Bylsma

CREATION OF VIRTUAL REALITY MODELING LANGUAGE (VRML) STATIONARY XY VIEW DATA FROM DISPLACEMENT DATA

Distribution: Approved for public release; distribution is unlimited.

U.S. Army Research, Development and Engineering Command
U.S. Army Tank-automotive and Armaments Research
Development and Engineering Center
Detroit Arsenal
6501 East 11 Mile Road
Warren, Michigan 48397-5000

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **31 DEC 2004**	2. REPORT TYPE	3. DATES COVERED **-**

4. TITLE AND SUBTITLE
CREATION OF VIRTUAL REALITY MODELING LANGUAGE (VRML) STATIONARY XY VIEW DATA FROM DISPLACEMENT DATA

5a. CONTRACT NUMBER
5b. GRANT NUMBER
5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
WESLEY BYLSMA

5d. PROJECT NUMBER
5e. TASK NUMBER
5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US ARMY TARDEC - NATIONAL AUTOMOTIVE CENTER,ATTN: AMSRD-TAR-N/MS157,6501 EAST 11 MILE RD,WARREN,MI,48397-5000

8. PERFORMING ORGANIZATION REPORT NUMBER
14386

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR’S ACRONYM(S)

11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
A process for convert body displacement (position/orientation) data to stationary XY view data, similar to a camera position moving in XY with the body, using the AWK programming language is presented, with the intent of future use in scene assembly into a Virtual Reality Modeling Language (VRML) file.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**9**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

1.0 INTRODUCTION....1

2.0 DISPLACEMENT FORMAT....2

3.0 VIEW FORMAT....2

4.0 VRML FORMAT....3

5.0 GAWK CONVERSION....5

5.1 INPUT....5

5.2 PARAMETERS....5

5.3 OUTPUT....6

5.4 CODE SECTIONS....6

6.0 SUMMARY/CONCLUSION....6

CONTACT....6

REFERENCES....6

DEFINITIONS, ACRONYMS, ABBREVIATIONS....7

APPENDIX A – DIS2Z0POS.AWK SCRIPT....9

FIGURE 1 – ESSENTIAL VRML VISUALIZATION COMPONENTS....1

TABLE 1 - DISPLACEMENT FILE VARIABLE FORMAT DEFINITION....2

TABLE 2 - VIEW FILE VARIABLE FORMAT DEFINITION....2

FIGURE 2 – GAWK PROCESS....5

TABLE 3 - PARAMETER DEFINITIONS....5

Technical Report 14386
December 2004

CREATION OF VIRTUAL REALITY MODELING LANGUAGE (VRML) STATIONARY XY VIEW DATA FROM DISPLACEMENT DATA

Wesley Bylsma
U.S. Army Research, Development and Engineering Command (RDECOM)
U.S. Army Tank-automotive and Armaments Research, Development and Engineering Center (TARDEC)
National Automotive Center (NAC)
ATTN: AMSRD-TAR-N/MS157
6501 E 11 Mile Road
Warren, Michigan 48397-5000

1.0 INTRODUCTION

Visualization of complex information is one of the best ways to communicate its meaning. The focus of this effort is on the creation of the view portion a Virtual Reality Modeling Language (VRML) file that is used to visualize ground vehicle simulations. As Figure 1 depicts, there are five essential elements that should be included within the composite VRML file for meaningful visualization effects. Only the view element is discussed here.

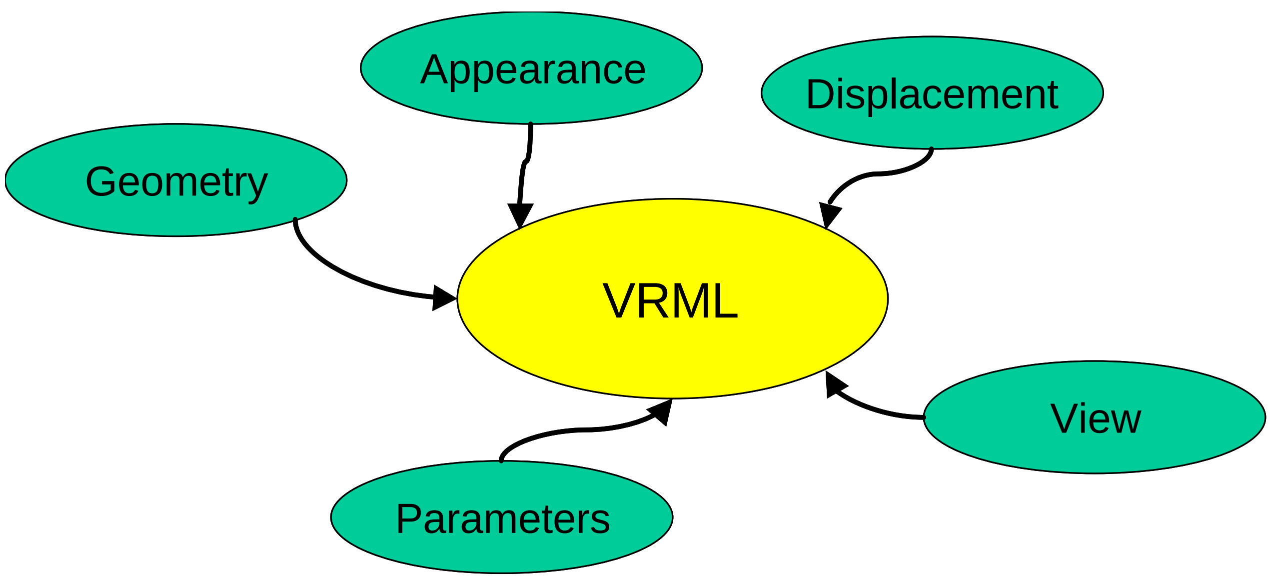

Figure 1 – Essential VRML Visualization Components

The view element is a generic file with the ".view" extension that contains position, orientation, and field of view information for specified geometry parts. This report addresses the conversion process between the displacement (".displacement") format to the generic stationary xy view (".view") format to be used with VRML. The conversion process is accomplished with the AWK Programming Language, named after its authors (Alfred V. Aho, Brian W. Kernighan, and Peter J. Weinberger at Bell Labs), which is designed to provide easy data manipulation and extraction of text files. In this case the Free Software Foundation's GNU version, GAWK, is used. More conversion processes may be developed in the future for inclusion of other view file formats. The focus of this discussion is restricted to the conversion of the displacement format. Section 2.0 begins with a discussion of the displacement format. Section 3.0 discusses the view format, section 4.0 discusses the VRML format that will be generated from the view format, and section 5.0 outlines the GAWK conversion processes with subsections on specific topics.

2.0 DISPLACEMENT FORMAT

The displacement file is a simple ASCII file. Its format is laid out into one section encapsulated by a header and trailer line. Below is an example ".displacement" file:

```
# A-HA1.BOD
#NSTEPS:   2226
1.003706 0.002024 0.645516 0.000000 -0.000010 0.000889 -0.000076
1.145413 0.002060 0.647116 0.000000 -0.000004 0.001006 -0.000096
1.287602 0.002087 0.649605 0.000000 0.000001 0.000977 -0.000118
1.429893 0.002399 0.651866 0.000000 0.000015 0.000858 -0.000203
...
0.441985 0.001879 0.653466 0.000000 -0.000007 -0.000388 -0.000085
0.581939 0.001908 0.650418 0.000000 -0.000012 -0.000144 -0.000066
0.722046 0.001944 0.647421 0.000000 -0.000015 0.000231 -0.000058
0.862559 0.001985 0.645643 0.000000 -0.000013 0.000611 -0.000062
#END
```

The first line is a comment denoting the body name the data is for. The section begins with "#NSTEPS:" and ends with "#END". Following the semicolon character in "#NSTEPS:" the number of time steps should be given. Each line between the header and trailer contains displacement information for each time step. For each time step the position, orientation, and scale can be specified. A variable format exists because displacement information may not need each of these three components. Table 1 defines how each type is specified and what nodes are used for them (NF = number of fields in gawk).

Table 1 - Displacement File Variable Format Definition

NF	Type	Nodes
>=3	Position	PositionInterpolator
>=7	Position/Orientation	PositionInterpolator/OrientationInterpolator
>=10	Position/Orientation/Scale	PositionInterpolator/OrientationInterpolator/PositionInterpolator

The above example file contains position and orientation (NF >= 7) displacement data, but not scale data. Note that orientation data requires position data, and scale data requires position and orientation data.

3.0 VIEW FORMAT

The view file has the same format as the displacement file, only with a few changes.. The only difference, besides some additonal comment lines, is the content for each time step. Below is an example ".view" file:

```
# A-TCH.BOD.view
# A-TCH.BOD
#NSTEPS:   2226
0.000449236 9.488932 0.001028 0
0.000898473 9.630410 0.001040 0
0.00134771 9.772396 0.001052 0
0.00179695 9.914128 0.001065 0
...
0.998203 8.925052 0.000976 0
0.999102 9.065768 0.000988 0
0.999551 9.206738 0.000999 0
1 9.347708 0.001013 0
#END
```

See a description of the format as described in section 2.0. For each time step the time fraction (percentage of total time), position, orientation, and field of view can be specified. A variable format exists because view information may not need each of these three components. Table 2 defines how each type is specified and what nodes are used for them (NF = number of fields in gawk).

Table 2 - View File Variable Format Definition

NF	Type	Nodes
>=4	Position	PositionInterpolator
>=8	Position/Orientation	PositionInterpolator/OrientationInterpolator
>=9	Position/Orientation/FieldOfView	PositionInterpolator/OrientationInterpolator/ScalarInterpolator

The above example file contains position (NF >= 4) view data, but not position or field of view data. Note that orientation data requires position data, and field of view data requires position and orientation data.

4.0 VRML FORMAT

The view data used in the VRML file [1] is included using the PositionInterpolator, OrientationInterpolator, and ScalarInterpolator nodes as defined in ISO 14772-1:1997. A TimeSensor node and ROUTE statement are also required. The creation of these nodes is done during final scene assembly with another program. A description is included here to help understand where the view data is included in the final VRML scene file.

The PositionInterpolator Node template is

DEF VPx PositionInterpolator { key ... keyvalue }.

This node defines the view position. By including a name to the node definition, it can be referenced later in the ROUTE statement which is based off the TimeSensor node to give motion. This nodes key values are sets of three floating point numbers---x,y,z values.

The OrientationInterpolator Node template is

DEF VOx OrientationInterpolator { key ... keyvalue }

This node defines the view orientation. By including a name to the node definition, it can be referenced later in the ROUTE statement which is also based off the TimeSensor node to give rotational motion. This nodes key values are sets of four floating point numbers. From [2], the first three values specify a normalized rotation axis vector about which the rotation takes place. The fourth value specifies the amount of right-handed rotation about that axis in radians. The 3x3 matrix representation of a rotation (x y z a) is

$$\begin{bmatrix} tx2+c & txy+sz & txz-sy \\ txy-sz & ty2+c & tyz+sx \\ txz+sy & tyz-sx & tz2+c \end{bmatrix}$$

where **c = cos(a)**, **s = sin(a)**, and **t = 1-c**.

The ScalarInterpolator Node template is

DEF VFx ScalarInterpolator { key ... keyvalue }.

This node defines the field of view. By including a name to the node definition, it can be referenced later in the ROUTE statement which is based off the TimeSensor node to give motion. This nodes key values are sets of floating point numbers.

An example section of a VRML (.wrl) file is included below:

```
DEF VP9 PositionInterpolator {
  key [
0.000449,
0.000898,
0.001348,
0.001797,
...
0.998652,
0.999102,
0.999551,
1.000000,
]
  keyValue [
9.488932 0.001028 1.366876,
9.630410 0.001040 1.366850,
9.772396 0.001052 1.366749,
9.914128 0.001065 1.366520,
...
8.925052 0.000976 1.366444,
9.065768 0.000988 1.366571,
9.206738 0.000999 1.366698,
9.347708 0.001013 1.366825,
] }
```

```
DEF VO9 OrientationInterpolator {
  key [
0.000449,
0.000898,
0.001348,
0.001797,
...
0.998652,
0.999102,
0.999551,
1.000000,
]
  keyValue [
0.008944 0.000474 0.003410 -0.000029,
0.008944 0.000474 0.003276 -0.000024,
0.008944 0.000471 0.003212 -0.000019,
0.008944 0.000465 0.003240 -0.000016,
...
0.008944 0.000429 0.003737 -0.000027,
0.008944 0.000447 0.003771 -0.000032,
0.008944 0.000460 0.003708 -0.000034,
0.008944 0.000469 0.003573 -0.000033,
] }
DEF VF9 ScalarInterpolator {
  key [
0.000449,
0.000898,
0.001348,
0.001797,
...
0.998652,
0.999102,
0.999551,
1.000000,
]
  keyValue [
0.785398,
0.785398,
0.785398,
0.785398,
...
0.785398,
0.785398,
0.785398,
0.785398,
] }
```

For each interpolator node there is an associated ROUTE statement that is based on the TimeSensor node. The ROUTE statement sets a routing path between the time change in the TimeSensor node to the interpolator node so that the view (position, orientation, or field of view) can be looked up. This "looked up" value is then routed to a Viewpoint node that changes the viewpoint as new values are routed to it. An example section of this part of the VRML (.wrl) file is included below:

```
DEF Tic TimeSensor { cycleInterval 73.33 loop TRUE }
...
Transform { translation 0 0 0 rotation 1 0 0 -1.57 children [
  DEF TG9 Transform { children [
  DEF V9 Viewpoint { position 0.000000 -10.000000 0.000000 orientation 1.000000 0.000000 0.000000 1.570000
description "vicplsROTbbody" }
    DEF S9_1 Shape {
      geometry USE C9_P1
      appearance USE A9_1
    }
    DEF S9_2 Shape {
      geometry USE C9_P2
      appearance USE A9_2
    }
    DEF S9_3 Shape {
      geometry USE C9_P3
      appearance USE A9_3
    }
    DEF S9_4 Shape {
      geometry USE C9_P4
      appearance USE A9_4
    }
```

```
  ] }
...
] }
...
ROUTE Tic.fraction_changed TO VP9.set_fraction
ROUTE VP9.value_changed TO V9.set_position
ROUTE Tic.fraction_changed TO VO9.set_fraction
ROUTE VO9.value_changed TO V9.set_orientation
ROUTE Tic.fraction_changed TO VF9.set_fraction
ROUTE VF9.value_changed TO V9.set_fieldOfView
```

In the above example, the view name will be "vicplsROTbbody" in the menu of views to select.

5.0 GAWK CONVERSION

The creation of the view portion of the VRML file is done with GAWK. This is a very useful scripting language that is available for UNIX and Windows operating systems from The Free Software Foundation ("www.gnu.org/software/gawk/gawk.html") or directly from Bell Labs ("cm.bell-labs.com/cm/cs/awkbook/"). Figure 2 outlines the AWK process.

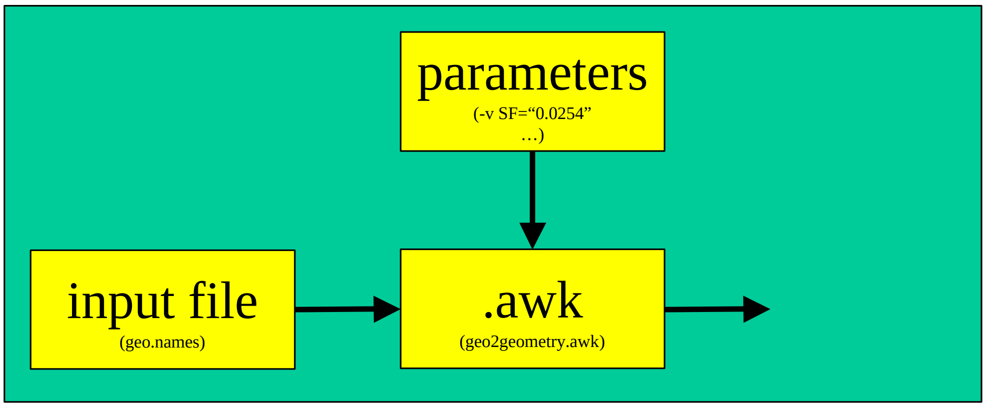

Figure 2 – GAWK process

Parameters are passed to the GAWK script before it begins processing the input file. Output can be sent to the standard output or a specified file. An example of the calling structure with the script name "dis2z0pos.awk" is given below

```
gawk -v SRC="./testd/" -v DES="./testd/" -f ../src/dis2z0pos.awk z0.names
```

5.1 INPUT

The input file is named "z0.names". Its contents are just a list of files to convert (1 column). The first column is the displacement (".displacement") file name. An example is included below:

```
A-TCH.BOD
A-TLOAD.BOD
```

Notice that the file extension is not included in the name.

5.2 PARAMETERS

Table 3 defines and describes the parameters passed to the GAWK script for processing the displacement files. The VRML file assumes all values are in SI units--meters for distance. Note that all parameters are passed as strings within double quotation marks.

Table 3 - Parameter Definitions

VARIABLE	DESCRIPTION	EXAMPLE
SRC	Source Directory.	"./geo/"
DES	Destination Directory.	"./testd/"

It is assumed that all file names listed in the "z0.names" input file have the same extension (in the example ".displacement").

5.3 OUTPUT

The converted file will have the same name as in the "z0.names" file, but with a ".view" extension. This format is discussed in section 3.0.

While it is not considered directly part of the conversion process, output of a formatted version of the "z0.names" file is required for scene assembly as discussed in Section 1.0 and Figure 1. An example of this simple GAWK program is given below

```
gawk 'BEGIN {cnt=0;}{if (substr($0,1,1)!="#"){cnt++;printf "%3d   %s\n", cnt,$1;}' z0.names > V.names
```

The "V.names" file is used for visual inspection of the view index number to ensure proper alignment with other VRML properties and is used in final scene assembly. An example of the "V.names" file format is given below.

```
  1   A-TCH.BOD
  2   A-TLOAD.BOD
```

5.4 CODE SECTIONS

The GAWK script "dis2z0pos.awk" is included in Appendix A. It begins with initializing variables. The position data for each displacement file is read in and written to each view file with the same name excluding the z data which is set to zero. Processing for each file name is as follows:

1. Set input (displacement) and output (view) file names
2. Find displacement file Header
3. Read position data (x,y,z) from displacement file
4. Wite position data (x,y,0) to view file
5. Write end of section marker to view file

6.0 SUMMARY/CONCLUSION

A simple script based conversion process between displacement and view format was described for use in scene assembly of VRML files. It should be noted that the scene assembly portion mentioned in section 1.0 could be done with X3D [2]. Currently, however, many advanced utilities, such as Cortona Movie Maker [3] will only work with VRML and therefore is the focus at this time.

Future view creation scripts will include orientation and field of view with the position data.

CONTACT

The author is an engineer with the U.S. Army Research, Development and Engineering Command (RDECOM), located at the U.S. Army Tank-automotive and Armaments Research, Development and Engineering Center (TARDEC). Interested parties can contact the author at the U.S. Army Tank-automotive and Armaments Research, Development and Engineering Center (TARDEC), ATTN: AMSRD-TAR-N/MS157, 6501 E 11 Mile Rd., Warren, Michigan 48397-5000, email: “bylsmaw@tacom.army.mil”.

REFERENCES

[1] The Virtual Reality Modeling Language (VRML), ISO/IEC 14772-1:1997 and 14772-2:2002, "www.web3d.org". (The Virtual Reality Modeling Language consists of two parts. Part 1 (ISO/IEC 14772-1) defines the base functionality and text encoding for VRML. Part 2 (ISO/IEC FDIS 14772-2) defines the base functionality and all bindings for the VRML External Authoring Interface).

[2] X3D, ISO/IEC Draft 19776-1:200x, 19776-2:200x, 19777:200x, "www.web3d.org". (X3D encodings — ISO/IEC FDIS (Final Draft International Standard) 19776-1:200x (XML encoding) (.html) (.zip 220KB) 2004-09-26 Specifies the encoding of X3D files using the Extensible Markup Language (XML). X3D encodings — ISO/IEC FDIS (Final Draft International Standard) 19776-2:200x (Classic VRML encoding) (.html) (.zip 90KB) 2004-07-21 Specifies the encoding of the functionality and constructs defined in X3D using Classic VRML encoding. X3D language bindings — ISO/IEC FCD (Final Committee Draft) 19777:200x (.html) (.zip 143KB) 2003-05-15 Specifies the binding of the services in the X3D architecture to the ECMAScript programming language for use in X3D internal representation (Script nodes) and for external application access Specifies the binding of the services in the X3D architecture to the Java programming language for use in X3D internal representation (Script nodes) and for external application access).

[3] Parallelgraphics, Inc. "www.parallelgraphics.com".

DEFINITIONS, ACRONYMS, ABBREVIATIONS
RDECOM – U.S. Army Research, Development and Engineering Center
TACOM - U.S. Army Tank-automotive and Armaments Command
TARDEC - TACOM Research, Development and Engineering Center
NAC - National Automotive Center
DADS - Dynamic Analysis and Design System
DIVA - DADS Interactive Visualizer

APPENDIX A – DIS2Z0POS.AWK SCRIPT

```
# dis2z0pos.awk
#
# -v SRC="./testd/"
# -v DES=".testd/"
#
BEGIN {
   cNSTEPS = "#NSTEPS:"
   cEND = "#END";
   mcnt = 0;
}
{
   fi = SRC""$1".displacement";
   fo = DES""$1".view";
   print "Processing ",fi,"->",fo;
   print "#",$1".view" > fo;
   while (err = getline < fi > 0)
   {
      print $0 >> fo
      if (substr($0,1,length(cNSTEPS)) == cNSTEPS)
      {
         nsteps = $2;
         cnt = 0;
         while (err = getline < fi>0 && substr($0,1,length(cEND)) != cEND)
         {
            cnt++;
            x[cnt] = $1;
            y[cnt] = $2;
            z[cnt] = $3;
            print cnt/nsteps, x[cnt], y[cnt], 0.0 >> fo;
         }
         print "#END" >> fo;
      }
      print "Done Reading",fi;
   }
   close(fi);
}
```